Diseño arquitectónico de una Aplicación

Programación de Aplicaciones Móviles Nativas

Autor: Fernando Sanfiel Reyes

Daniel Betancor Zamora

Pablo Santana Susilla

Fecha: 08/10/2023

Índice

1.- Definición del dominio

Nuestra aplicación tiene, como principal objetivo, facilitar a los usuarios la gestión de sus compras. Proporcionando, gracias a su funcionalidad principal, la información nutricional específica de cada alimento mediante el escaneo del código de barras único, a través de la cámara fotográfica del dispositivo móvil del usuario. Además, esta funcionalidad está especialmente relacionada con la creación y gestión de listas de alimentos y otros productos disponibles en el supermercado, los cuáles el usuario puede ir añadiendo y eliminando, incluso, teniendo la posibilidad de compartir las listas con otros contactos para dividir los gastos de dicha lista. De este modo, con precios aproximados, el usuario puede hacer un seguimiento de sus gastos diarios, semanales o mensuales en lo que a compras alimenticias se refiere.

A continuación, se mostrará de forma más detallada todas aquellas funcionalidades que acaban de ser comentadas, y en las cuales se basa nuestra propuesta de proyecto, pudiendo observar de manera más clara el dominio de la aplicación.

CREACIÓN DE LISTAS DE LA COMPRA

- Permitir a los usuarios crear listas de productos de supermercados de manera intuitiva y personalizada.
- Facilitar la adición y eliminación de elementos de la lista.
- Mejorar la experiencia de compra al ayudar a los usuarios a recordar todos los productos necesarios y evitar olvidos.

COMPARTIR LISTAS DE LA COMPRA

- Posibilitar que los usuarios compartan sus listas de compra con familiares, amigos o compañeros de piso.
- Permitir la división de gastos entre usuarios, lo que facilita la gestión de presupuestos compartidos y la planificación de compras colaborativas.

ESCANEO DE CÓDIGOS DE PRODUCTOS

- Integrar una función de escaneo de códigos de barras que brinde información detallada sobre los productos, como la información nutricional y los precios.
- Facilitar la adición de los productos escaneados a las listas de la compra creadas o a favoritos (para guardar aquellos productos más usados por el usuario).

INFORMACIÓN NUTRICIONAL DETALLADA

- Proporcionar acceso rápido y sencillo a la información nutricional de los productos.
- Ayudar a los usuarios a tomar decisiones informadas sobre sus compras, promoviendo hábitos alimenticios saludables.

CONTROL DEL GASTO

- Registrar automáticamente los gastos de cada compra, permitiendo a los usuarios llevar un seguimiento detallado de sus gastos diarios, semanales y mensuales en el supermercado.
- Generar informes y estadísticas para ayudar a los usuarios a gestionar su presupuesto y controlar sus gastos de manera efectiva.

2.- Identificación de los principales elementos de la arquitectura elegida

En el caso de nuestra aplicación, hemos creído conveniente utilizar la ***Clean Architecture***, por lo que, a continuación, se comentarán los principales aspectos de esta arquitectura que han sido identificados en nuestra aplicación, y de qué manera serán implementados en nuestro proyecto.

Concretamente, se identificarán los elementos de nuestra aplicación correspondientes a cada una de las capas de la *Clean Architecture*, entre las cuáles se encuentra la **Capa de Dominio** (Domain Layer), la **Capa de Aplicación** (Application Layer), la **Capa de Interfaces** (Interface Layer), y la **Capa de Infraestructura** (Infrastructure Layer).

CAPA DE DOMINIO (DOMAIN LAYER)

- **Reglas de Negocio**: Aquí definiríamos las reglas fundamentales de negocio de GoShop, como las reglas para evitar productos duplicados en una lista de compra, cómo calcular el gasto total y las reglas para compartir listas de compra entre usuarios.
- **Entidades de Dominio**: Creamos clases que representan conceptos clave en la aplicación, como *ListaDeCompra*, *Producto*, *Usuario*… Con atributos y métodos relacionados con sus propiedades y comportamientos.
- **Validaciones**: Implementaríamos validaciones para garantizar que los datos sean coherentes y cumplan con las reglas de negocio, como asegurarse de que un producto tenga un nombre válido y que la cantidad no sea negativa.

CAPA DE APLICACIÓN (APPLICATION LAYER)

- **Casos de Uso**: Definimos los casos de uso que representan las acciones que los usuarios pueden realizar, como *CrearListaDeCompra*, *CompartirLista*, *EscanearCodigoDeBarras*, *CalcularGasto*. Cada caso de uso contiene la lógica para llevar a cabo la acción correspondiente.
- **Interacción con la Capa de Dominio**: Los casos de uso se comunican con la capa de dominio para aplicar las reglas de negocio definidas y garantizar que se mantenga la coherencia de los datos.

CAPA DE INTERFACES (INTERFACE LAYER)

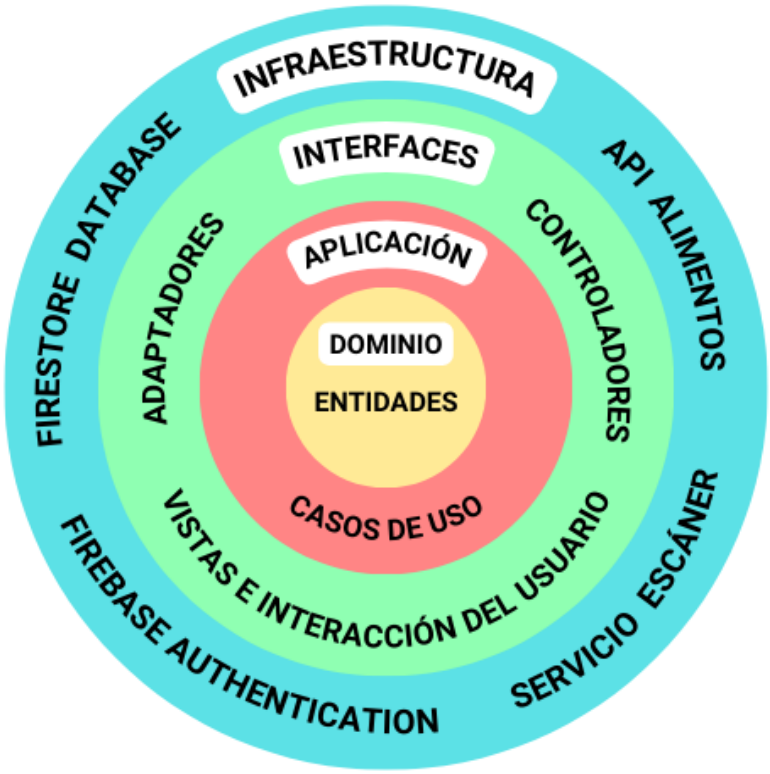
- **Interfaces o Puertos**: Definimos interfaces que describen cómo la aplicación interactuará con el exterior, como la interfaz de usuario y otros servicios externos. Por ejemplo, tendríamos un puerto de escaneo de productos para interactuar con el escáner de códigos de barras y un puerto de listas de compra para gestionar listas de compra.
- **Manejo de Excepciones**: Implementamos manejo de excepciones para gestionar errores y excepciones que puedan ocurrir en la aplicación, como errores de validación o problemas de conexión.

CAPA DE INFRAESTRUCTURA (INFRASTRUCTURE LAYER)

- **Adaptadores para Bases de Datos y Servicios Externos**: Implementamos adaptadores que conectan la aplicación con recursos externos. Por ejemplo, un adaptador de base de datos Firebase para el almacenamiento de listas de compra, usuarios, gastos…
- **Acceso a Datos**: La capa de infraestructura se encarga de gestionar el acceso a bases de datos y servicios externos, como almacenar y recuperar listas de compra en Firebase o interactuar con una API de información nutricional a través de la red.
- **Gestión de Almacenamiento de Datos**: Aquí definimos cómo se almacenan los datos, ya sea en una base de datos local, en la nube o en otro sistema de almacenamiento.

3.- Diseño de la Arquitectura

En este apartado, presentaremos un diagrama que ilustra la estructura y organización de la *Clean Architecture* aplicada a nuestra aplicación, **GoShop**. Como ya hemos visto, la *Clean Architecture* es un enfoque de diseño de software que enfatiza la separación de preocupaciones y la modularización del código, lo que facilita la escalabilidad, el mantenimiento y la prueba del software.



4.- Casos de Uso

En esta sección, presentaremos dos casos de uso fundamentales de la aplicación GoShop. Los casos de uso son descripciones detalladas de las funcionalidades clave que los usuarios pueden realizar en la aplicación. Cada caso de uso se centra en una tarea específica y proporciona una comprensión clara de cómo los usuarios interactúan con GoShop para lograr sus objetivos.

<table>
<tr><td>CU - 01</td><td>Leer código de barras</td></tr>
<tr><td>Precondición</td><td>El usuario se deberá de haber previamente registrado en la aplicación y debe querer conocer los detalles de un alimento.</td></tr>
<tr><td>Descripción</td><td>Un usuario que desea conocer los detalles de un alimento utiliza la cámara de su móvil para leer el código de barras</td></tr>
<tr><td>Secuencia</td><td>
<table>
<tr><th>Paso</th><th>Acción</th></tr>
<tr><td>1</td><td>El usuario desea conocer los detalles de un alimento.</td></tr>
<tr><td>2</td><td>El usuario abre el apartado de cámara dentro de la aplicación.</td></tr>
<tr><td>3</td><td>El usuario apunta con la cámara al código de barras del alimento.</td></tr>
<tr><td>4</td><td>Se hace una consulta a la API a través del código escaneado y se devuelve la información correspondiente.</td></tr>
<tr><td>5</td><td>La aplicación muestra los detalles del alimento leído.</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>El usuario lee los detalles del alimento deseado.</td></tr>
<tr><td>Excepciones</td><td>
<table>
<tr><th>Paso</th><th>Acción</th></tr>
<tr><td>5</td><td>Si el alimento no se encuentra registrado en la aplicación.
<table>
<tr><td>E1.</td><td>El sistema informa que el alimento leído no se encuentra en la base de datos</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>CU - 02</td><td colspan="2">Crear una lista</td></tr>
<tr><td>Precondición</td><td colspan="2">El usuario se deberá de haber registrado previamente en la aplicación y debe querer crear una lista de la compra.</td></tr>
<tr><td>Descripción</td><td colspan="2">Un usuario que quiera anotar qué alimentos comprar, debe poder crear una lista de la compra.</td></tr>
<tr><td>Secuencia</td><td colspan="2"><table>
<tr><th>Paso</th><th>Acción</th></tr>
<tr><td>1</td><td>El usuario desea anotar qué alimentos comprar en su próxima compra.</td></tr>
<tr><td>2</td><td>El usuario abre el apartado donde se pueden ver y crear listas de la compra.</td></tr>
<tr><td>3</td><td>El usuario crea una nueva lista de la compra.</td></tr>
<tr><td>4</td><td>El usuario le pone nombre a la nueva lista.</td></tr>
<tr><td>5</td><td>Los alimentos anotados quedan guardados en la lista.</td></tr>
</table></td></tr>
<tr><td>Postcondición</td><td colspan="2">El usuario puede consultar los alimentos anotados.</td></tr>
<tr><td>Excepciones</td><td colspan="2">No existe ninguna excepción.</td></tr>
</table>

5.- Conclusión

La elección de esta arquitectura ha estado basada principalmente en su sencilla **escalabilidad**, ya que nuestra aplicación en un futuro podría añadir lectura mediante código QR a modo de funcionalidad extra o sustitución del código de barras. Además de esto, también podríamos encontrar una mejor base de datos que *Firebase* o, incluso, poder llegar a acuerdos con empresas dedicadas a la venta de alimentos para que pudieran realizar publicación de sus ofertas o actualizaciones y, haciendo uso del concepto de los adaptadores, serían funciones fácilmente implementables sin tener que modificar la estructura principal de la aplicación.

En comparación con el modelo **MVVM** la *Clean Arquitecture* es más apropiada, ya que el modelo MVVM desacopla las funcionalidades del dominio de negocio y en nuestro caso hay funcionalidades principales que son relevantes para nuestro modelo de negocio.

En el caso de comparación con el **MVP** se sobrepone la *Clean Arquitecture* debido a que el uso del *Presenter* como intermediario no es tan optimo ni escalable como el uso de los *Adapters* que se encuentra en la *Clean Arquitecture*.

La comparación con el modelo **MVI** es bastante similar con la del modelo MVP, ya que la función del *Intent* en el modelo MVI, se asemeja a la función del *Presenter* en el modelo MVP. Teniendo lo anterior en cuenta, se considera mejor opción la *Clean Arquitecture* debido al uso de los *Adapters* debido a que le proporciona mayor escalabilidad a nuestra aplicación.

6.- Enlace a GitHub

Repositorio en GitHub: https://github.com/PAMN2023/ArchitecturalAppDesign.git

7.- Bibliografía

Andalucía, J. d. (04 de 11 de 2022). *Guía para la redacción de casos de uso*. Obtenido de Guía para la redacción de casos de uso: https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/416

Delespierre, B. (04 de 10 de 2021). *Clean Architecture with Laravel*. Obtenido de Clean Architecture with Laravel: https://dev.to/bdelespierre/how-to-implement-clean-architecture-with-laravel-2f2i

Soni, S. (16 de 03 de 2018). *MVI(Model-View-Intent) Pattern in Android*. Obtenido de MVI(Model-View-Intent) Pattern in Android: https://medium.com/code-yoga/mvi-model-view-intent-pattern-in-android-98c143d1ee7c

Wikipedia. (07 de 07 de 2021). *Modelo–vista–modelo de vista*. Obtenido de Modelo–vista–modelo de vista: https://es.wikipedia.org/wiki/Modelo–vista–modelo_de_vista

Wikipedia. (15 de 04 de 2022). *Modelo–vista–presentador (MVP)*. Obtenido de Modelo–vista–presentador (MVP): https://es.wikipedia.org/wiki/Modelo–vista–presentador